

# Emulating a large memory sequential machine with a collection of small memory ones

James Hanlon, Simon J. Hollis, and David May

**Abstract**—Sequential computation is well understood but does not scale well with current technology. Within the next decade, systems will contain large numbers of processors with potentially thousands of processors per chip. Despite this, many computational problems exhibit little or no parallelism and many existing formulations are sequential. Therefore, it is essential that highly parallel architectures can support sequential computation by emulating large memories with collections of smaller ones, thus supporting efficient execution of sequential programs or sequential algorithms included as part of parallel programs. This paper presents a novel tiled parallel architecture which can scale to thousands of processors per-chip and can deliver this ability. Provision of an interconnect with scalable low-latency communications is essential for this and the realistic construction of such a system with a high-degree switch and a Clos-based network is presented. Experimental evaluation shows that sequential programs can be executed with only a factor of 2 to 3 slowdown when compared to a conventional sequential machine and that the area is roughly only a factor of two larger. This seems an acceptable price to pay for an architecture that can switch between executing highly parallel programs and sequential programs with large memory requirements.

**Index Terms**—Parallel computing, many-core, network-on-chip, memory system, distributed memory, message passing, DRAM, Clos network.

## I. INTRODUCTION

SEQUENTIAL COMPUTATION is based on the abstraction of a single, large randomly-accessible memory in which data structures are manipulated. By scaling the size of the memory, a machine can handle arbitrarily large data sets and programs, and operation throughput can be scaled by increasing the execution frequency. This model is expressive and well understood but does not scale well with current technology: growth in single processor performance has slowed dramatically and memory access latency is several orders of magnitude larger than the time taken to execute basic arithmetic operations. Parallelism is now the primary means for sustaining growth in computational performance [1, p. 109] and it looks increasingly certain that future systems will involve large numbers of processors, potentially with thousands of processors per-chip [2], [3].

Exploitation of highly parallel systems requires programs that exhibit large amounts of parallelism and it is the challenge of parallel computing research to discover means of decomposing and expressing problems in this way. Despite this, problems which exhibit little or no parallelism remain important

and the number of programs and algorithms formulated in this way is large. Therefore, it is essential for the broad adoption of highly parallel architectures that they can efficiently support sequential programming techniques. Current parallel systems tackle this problem by over-provisioning the amount of memory available per-processor, typically by providing access to a monolithic DRAM (dynamic random access memory) system. This is the approach of the the Intel MIC [4], Intel SCC [5], Tiler Tile [6] and IBM BlueGene [7] chip architectures, and many others follow the same approach. This works well for sequential computations but compromises parallel ones as access contention to the memory increases and the ability to exploit locality in the computation is impaired with a flat global address space. *Heterogeneous* architectures employ different types of processors to support different workloads and many new processors have graphics processing units (GPUs) that can support limited forms of parallelism, as well as CPUs geared towards serial workloads, such as ARM's *big.LITTLE* technology [8] and AMD's Accelerated Processing Units [9]. When it is efficient to do so, parallel tasks can be offloaded to the GPU, and the CPU can take care of other forms of parallelism and sequential work. The problem with this is that as understanding of parallelism develops and workloads change, then so must the degree of heterogeneity. These changes will be reflected in the programming model and this makes it difficult or impossible for existing programs to take advantage of improvements in future architectures.

A *homogeneous* system provides a better balance over all workloads as they change using a single portable programming model. To support sequential approaches with large memory requirements, rather than provisioning large amounts of memory per-processor, large memories can be emulated with collections of smaller ones. By scaling the number of processors, arbitrarily large memories can be emulated, although subject to the same physical and technological constraints of a single processor with a large memory. The result is a simple, flexible architecture where the amount of memory per processor can be increased to execute sequential or less-parallel programs. There is an overhead associated with this flexibility, and execution will be slower than a sequential machine, but the emulation can be made efficient so that the slowdown is low. Most importantly, the system maintains its ability to execute parallel programs efficiently.

Conventional sequential machines employ monolithic DRAMs to provide access to a large memory space. These are constructed as a collection of smaller DRAM arrays connected by an interconnect and are specialised to provide sequential random access. This incurs a latency overhead that is deter-

All authors are with the University of Bristol, Department of Computer Science, Merchant Venturers Building, Woodland Road, BS8 1UB, Bristol, UK  
E-mail: {hanlon, simon, dave}@compsci.bristol.ac.uk

mined by the structure and properties of the network, such as topology and the critical path of components through which data pass. In a parallel system, the network is able to support more general communication patterns and consequently access to different memories incurs a larger latency overhead, due to the same aspects of the network. Therefore, in order to deliver an efficient *emulation of a large memory*, it is essential that the overhead of communication is low and to use a network topology that has a low diameter.

This paper presents a highly parallel architecture with a Clos-based interconnect that can deliver an efficient emulation of a large memory sequential computation. Efficient is defined in this context as a low overhead compared to the conventional implementation of a sequential machine in terms of both *time* and *implementation cost*. Clos networks are chosen because their low diameter and regular recursive structure with fixed-degree nodes makes them attractive to build. This work contributes to the state-of-the-art by demonstrating that it is practical to connect large numbers of processors in this way on-chip using high-degree switches, and that such a system can be used to emulate a sequential machine to execute sequential programs efficiently. The proposed implementation is based on a reasonable set of assumptions relating to on-chip construction with DRAM memory integrated using 3D stacking. Experimental results obtained by simulation show that the overhead of additional processors and interconnect is comparable and scales well relative to the area occupied by a specialised monolithic memory, and the slowdown of sequential execution is low compared with a conventional sequential machine. This slowdown seems an acceptable price to pay for an architecture that can switch between executing highly parallel programs and sequential programs with large memory requirements.

The following specific contributions are made:

- 1) A proposal for a new parallel architecture that can scale to thousands of cores on a chip, based on a realistic VLSI model.
- 2) A simple scheme for this architecture to execute sequential computations with large memory requirements.
- 3) Analysis of the implementation cost of the system and how this scales in terms of the processing, interconnect and memory components, compared to a sequential machine.
- 4) Analysis of the proportion of memory accesses in general sequential programs to show that, in practice, accesses to large data structures constitutes 10% to 20% of executed operations.
- 5) Experimental results demonstrating that the execution of a conventional sequential program that manipulate large data structures can be executed by the proposed architecture at 25% to 50% of the throughput of a sequential machine, even when the data are distributed over thousands of processors.

The rest of this paper is organised as follows. Section II overviews related work; Section III presents the proposed parallel system and the emulation scheme; Section IV describes the area and performance models and their parameters used

in the evaluation; Section V describes the evaluation methodology and assumptions, and presents the experimental results; Section VI discusses potential engineering optimisations that could be made, the scalability of the proposed system with current technology and how future technologies may impact on this, and possible developments and extensions to the proposal; Section VII concludes.

## II. RELATED WORK

The need for scalable general-purpose systems has led to a number of proposals for explicitly parallel tiled architectures, where a system is built as a regular arrangement of processor-memory pairs connected by an interconnection network. This allows memories to be more tightly bound to processors and local access latency to be reduced. Prominent examples include the MIT Raw [10] and descendant Tile [6] architectures, Adapteva's Epiphany architecture [11] and Smart Memories [12], but despite an established theory of general-purpose parallel computation which requires low-diameter high-capacity networks [13], [14], [15], these systems neglect this theory and typically employ a 2D mesh interconnect topology, where communication latency scales linearly with the number of tiles and programs must be carefully mapped to preserve locality to obtain good performance. Where the class of programs is limited to a specific set of domains, a mesh interconnect can be effective, such as with the AsAP [16] and PicoChip [17] architectures for digital signal processing. In contrast, the INMOS transputer [18] and descendant XMOS XS1 architecture [19] were designed to support general-purpose parallelism with high performance networks [20], [21].

Clos [22] and the related fat-tree [13] networks have been widely studied and used for interconnecting parallel systems. They are well established as interconnects for large systems comprising thousands of processors in supercomputers such as the Connection Machine 5 [23], IBM's Roadrunner [24] and the SGI UV architecture [25], and in data centres as they can be built from commodity network equipment [26]. For these networks, high-degree switches significantly reduce the total switch requirement and the resulting diameter. The scaling of a switch is limited primarily by the delay and area requirements of the central crossbar component. A number of high-degree designs from  $32 \times 32$  to  $128 \times 128$  have been proposed and shown to be effective, e.g. [20], [27], [28]. With an increasing level of parallelism on-chip, various proposals for on-chip electrical e.g. [29], [30], [31] and optical, e.g. [32], [33] Clos/fat tree networks have been made to reduce latency, improve capacity and simplify programming, but in general these only consider relatively small systems up to 64 processors and low-degree  $4 \times 4$ ,  $8 \times 8$ , or  $16 \times 16$  crossbar switches.

This paper assumes the availability of 3D silicon integration of DRAM memory since it is the most practical way to integrate many small memories with processors as commodity DRAM is produced on a specialised process and cannot be integrated directly with microprocessor logic. Static RAM (SRAM) and DRAM embedded in a commodity process are 7 to 15 times less dense than contemporary commodity

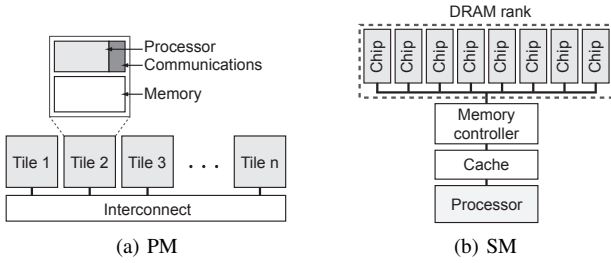


Fig. 1. Simplified organisation of the (a) proposed parallel architecture where a collection of processor-memory tiles are connected together by an interconnect to allow communication between all tiles; and the (b) conventional sequential architecture where a processor accesses a large memory via a memory controller, which consists of many small DRAM arrays, distributed over a collection of chips. A cache is used to reduce the number of accesses to this.

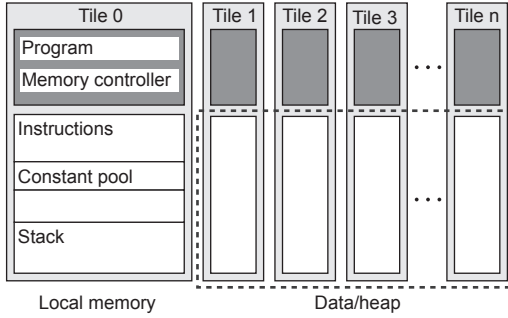


Fig. 2. Emulation of a sequential machine on the proposed parallel machine. A sequential process is executed by a single processor and its local memory contains the program, constant pool and stack. Global data is distributed over a collection of other tiles, managed by a memory controller process. This receives memory accesses from the program and directs them to the tile containing the associated address. Global memory is distributed over tiles 1 to  $n$  for clarity but tile 0 could also provide storage for global memory.

DRAM, and not a practical means of building systems with large amounts of memory. It is likely in the near future that this will be widely used to increase memory device density; stackable DRAMs are already commercially available from Tezzaron [34] and standards are emerging for embedded systems with Wide I/O [35] and in the desktop and server space with the Hybrid Memory Cube [36].

### III. PROPOSED SYSTEM

The proposed *parallel machine* (PM), as well as being able to execute parallel programs, can deliver an efficient emulation of a *large memory sequential machine* (SM), and therefore efficiently execute sequential programs. The PM consists of a set of *tiles* and each tile contains a processor, memory and communications system. This arrangement allows close physical proximity of the memory to processors to reduce access latency due to wire delay. This is essential for scaling performance in highly parallel programs and allows a simple processor architecture to be used as memory access time can be close to that of regular operations. Communication links are used to connect tiles to switches and a network topology is formed by connecting switches to each other. Processors can communicate with one another by passing messages on the interconnect. A larger system may contain multiple chips

and communication links can be connected between tiles on different chips, in the same way they are connected on-chip. This provides a uniform and efficient means of communicating throughout the system. Fig. 1a illustrates this organisation.

#### A. Emulating a sequential machine

A conventional SM is comprised of a processor attached to a memory system via a memory controller. A separate automatically managed cache memory is used to reduce the number of accesses to this. The main memory system is built as a collection of small DRAM arrays which are sized to trade-off well between density, latency and energy, and interconnected with a specialised network to transmit control signals, addresses and data [37]. These are typically distributed over a set of chips, called a *rank*, and the chips connect directly to the memory controller to provide a 64-bit access width. Fig. 1b illustrates this organisation. The principal way in which it differs from the PM is the amount of memory available to a processor.

Each tile of the PM has the capability to execute sequential programs, but to be able to execute arbitrary sequential programs it must also support those with large memory requirements. In a SM, memory accesses are usually divided between local and global storage. Access to global storage in the PM is provided with a collection of processors and their memories. This is managed by a software memory controller process which receives access requests in a contiguous address range and distributes them over the array of processors by sending messages on the interconnect. Typically, the requirements of local storage, which includes program, constant values and the stack, are small and could fit into a processor's local memory. If they exceeded this capacity, the program or stack could be split between processors, at the cost of some additional latency, and invoked or accessed remotely by passing messages<sup>1</sup>. Alternatively, a smaller emulated memory structure could be used. In this work, only the simple case is considered. Fig. 2 illustrates the separation of the memory address space over tiles and the processes executed in the system. This is essentially the mapping of computational and memory components of the SM to the PM (Fig. 1b to Fig. 1a). To access the emulated memory, instead of executing a conventional load or store instruction, a short request-reply communication sequence with the memory controller process is performed. Generation of these sequences can be integrated into compilation so that loads and stores to global memory are automatically directed at the emulated memory. This allows conventional sequential programs, and in particular legacy applications, to be compiled directly to this architecture. This is demonstrated in Section V-B4.

The performance of the SM emulation is determined by two components, the frequency of instruction execution and the latency and bandwidth performance of the memory system. Instruction throughput is increased by increasing the processor's clock frequency and by exploiting instruction-level parallelism

<sup>1</sup>The Adapiva Epiphany architecture [11] directly supports this concept by providing a configurable DMA engine which allows instructions or data to be fetched from remote cores.

(ILP) to increase the number of instructions executed per cycle. This is discussed in Section III-D. Bandwidth in the PM can match that of the SM by a combination of widening data paths, transmitting data serially at higher rates or by compressing it before transmission. Latency is more difficult to scale due to intrinsic physical delays, and hence is the focus of this work. This is reflected in historical DRAM performance scaling where in the time for bandwidth to double, latency improves only by around 20% to 40% [38]. In the PM, local memory accesses will be fast but remote access will incur additional latency due to delay through processors, switches and links. Minimising this is crucial for the PM to deliver an efficient emulation, and choosing a network topology with a small diameter compared to the number of nodes it connects is the principal way to achieve this.

### B. Switch

A high-degree crossbar switch, where all inputs can be simultaneously routed to their outputs is used as the basic network building block since it allows a variety of network topologies to be constructed, and multiple tiles can be attached to each switch. For a small number of tiles, a fully-connected network via a single switch is sufficient but for larger systems, switches are interconnected with links to form a network to provide communication between all tiles. By connecting  $k$  tiles to a single switch, the diameter of the network can be reduced by a factor of  $k$ . With a  $32 \times 32$  crossbar switch it is practical to connect up to half the links to tiles, leaving sufficient bandwidth and connectivity in the set of remaining links.

Following the design of the C104 [20] and XS1 [19] devices, the switch is wormhole-routed to minimise latency and support streamed or packetised communications. A route is opened through the switches in the network by a header *token* so that subsequent tokens experience no routing delay. Data can be streamed in this way or packetised by closing the route after the payload has been sent. For networks such as the Clos, hypercube and shuffle with appropriate routing strategies, little buffering is required [15]; for each inbound and outbound link around one packet (one word) is sufficient [39].

### C. Network topology

Clos networks are investigated in this work because they have a low-diameter, a regular recursive structure and can be constructed with fixed degree switches. In contrast, hypercubes require different sized switches for different network sizes and have complex overlapping connections. Compared to butterflies, Clos networks allow flexible provisioning of bandwidth in different stages, can exploit traffic locality between them and have non-blocking properties. A Clos network is an *indirect multi-stage* network that uses additional switches to make it more highly connected. A *folded* arrangement [39] reduces the switch requirement and is also known as a *fat tree* [13]. With  $p$  processors, the resulting diameter of the network is  $O(\log_k p)$ . Fig. 3 illustrates the  $32 \times 32$  switch topologies for 64, 256 and 1024 tile networks; the physical layout is discussed in Section IV-A.

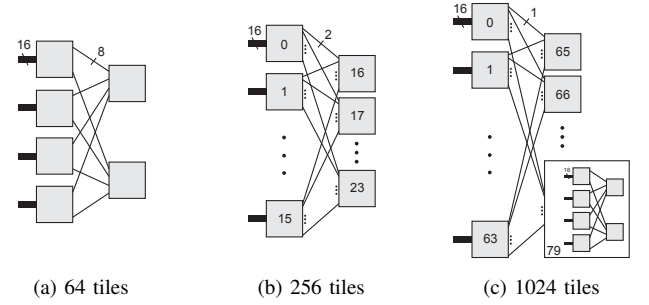


Fig. 3.  $32 \times 32$  crossbar switch topologies for various size of Clos networks. In each, all edge switches connect to 16 tiles. In the 1024-tile network, the internal switches are themselves constructed as a small Clos network to provide enough links, making the network 3-stage. The total bandwidth between the processors is maintained in each stage of the network.

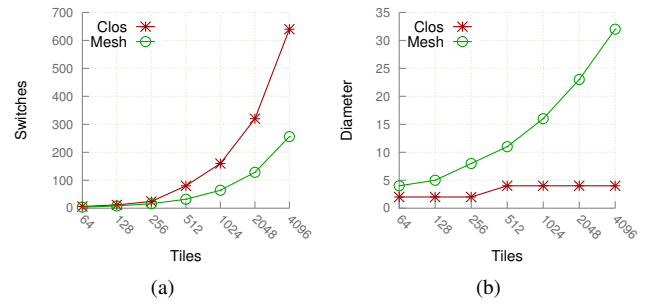


Fig. 4. Log-linear plots showing how the number of switches (a); and network diameter (b) scales as system size increases for Clos and 2D mesh networks built with  $32 \times 32$  crossbar switches.

A 2D mesh is used as a baseline comparison because of its widespread use as an interconnect topology. It is a simple structure that is appealing as it can be easily packaged on a 2D surface. Each node is connected to 4 others, except at the edges where links remain unconnected, resulting in a diameter of  $O(p)$ . In *torus* networks these edge links are ‘wrapped-around’, but this does not offer a significant improvement in diameter. A 2D mesh is much less highly connected than the Clos which reduces the cost of implementation but significantly limits its scalability. Fig. 4 illustrates this by showing how the number of switches and the diameter of both networks scale. The switch requirement scales more steeply for the Clos by a logarithmic factor but the resulting diameter is small and grows logarithmically in  $p$ . The evaluation of area in Section V-A shows that despite this the implementation cost, even for large systems, is not prohibitive.

### D. Processor

In principle any processor could be used. Complicated superscalar designs can deliver good sequential performance by exploiting ILP but they have significant area requirements and in the case of true sequential code with tight instruction dependencies, their performance will degenerate to a baseline without the benefits of the architectural optimisations. A simple architecture is necessary to integrate large numbers of processors on a chip and can reasonably be expected to deliver similar performance to a superscalar design for true

sequential programs and to within a factor of 3 to 4 for ones with ILP, based on the potential speedup that could be obtained with multiple execution units. Importantly though, collections of simple processors have the ability to exploit parallelism to scale performance far beyond what a single superscalar processor can offer. The choice of clock speed is dependent on the power and thermal constraints of the system.

The only requirements for the processor in this architecture are that it must be able to handle concurrent input and output to prevent deadlock from occurring. This can be achieved with a conventional interrupt system or multi-threading. To minimise communication latency, any overhead associated with performing communication must be low. Ideally, local communications complete in a similar time to conventional loads, stores, branches and arithmetic. One way to achieve this is to use dedicated registers or buffers as their fast access time avoids the overhead of a direct memory access system required for main memories with large access latencies. A 15-fold improvement in latency was observed by doing this in the Intel SCC [5].

#### IV. CHIP MODEL

In order to evaluate the efficiency of the emulation in terms of implementation cost and performance, it is necessary to develop a detailed model of how the system is built. This section presents models for area and latency based on a VLSI implementation of the system. The model uses current technology to produce a realistic design and in the evaluation, both the parallel and sequential systems are built in this way.

A system is packaged as a set of chips. The *processing chip* contains processor cores and the interconnect, and one or more commodity DRAM *memory chips* are integrated by stacking them on top and connecting between them with Through-Silicon Vias (TSVs). This is not possible with current production processes, which are limited to stacking a single DRAM die on a logic one and the largest capacity stacks are 0.5GB [34], but it is expected that stacks with multiple DRAM dice will be available in the next few years. This assumption allows a system with a larger memory capacity to be hypothesised, that is comparable to typical capacities of conventional DRAM systems where chips are packaged in dual inline memory module (DIMM) cards.

Each memory chip contains a number of separate DRAM memories with each memory connected directly to the corresponding processor tile between chips. In systems where memory area is larger than the processing area, a larger chip size for both systems could be chosen to accommodate the memory, or it could be divided between one or more chips. The chip model is based on a 28nm logic process and a 40nm DRAM process which is representative of current commodity devices. Parameters for the processing chip are summarised in Tab. I and discussed in Section IV-B.

For the processing chip, the XMOS XCore processor [19] is used as it provides direct support for multi-threading, message passing communication and has simple predictable behaviour and a low implementation cost. For chip wiring, it is assumed that all inter-switch wires are implemented in semi-global

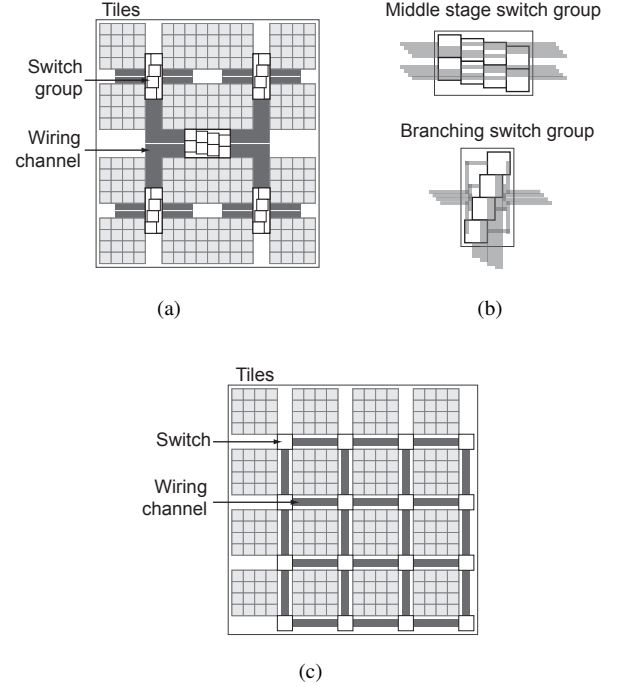


Fig. 5. Example chip layouts for 256-tile folded Clos (a) and 2D mesh (c) networks. The Clos network is based on a H-tree organisation, with groups of switches at each node organised in staggered sets to minimise area (b). Connections from switches to processors are not included.

metal layers with standard repeated wires and repeaters are inserted optimally to minimise latency. Wires with multi-cycle delays are pipelined and all wires are half shielded to reduce crosstalk. Tiles and switches are not placed directly underneath interconnect wires to simplify the insertion of repeater banks and minimum wire pitch is doubled to account for contacted wire pitch and to mitigate capacitance. It is assumed that 4 metal layers are available for routing global wires. As wiring channels are dedicated, all of these are used to route all inter-switch wires.

##### A. Layouts

The area requirement and wire delays for the Clos network are estimated using a layout based on a H-tree. In this, the middle stage core switches are placed in the centre of the chip and the next stage of switches is divided into four groups and each are placed in the centre of a quadrant surrounding the core switches. Connections are then made from all of the core switches to each of the next stage switches. This process continues recursively in each quadrant for each additional stage. An example layout for a 2-stage 256-tile folded Clos network relating to the topology of Fig. 3b is given in Fig. 5a.

Groups of switches are arranged in staggered sets to minimise the resulting size of their bounding box. This is constrained by the pitch of connecting wires in either the vertical or horizontal direction, which are routed on the 4 available metal layers, two for each direction. It is assumed that the resulting longer dimension of a group can extend into the wiring channel. The core stage switches only make



Parameter	Value
Process node	28nm
Semi-global interconnect wire pitch	125nm
Peripheral circuitry area overhead	20%
Processor core area	0.10mm <sup>2</sup>
Switch area	0.05mm <sup>2</sup>
Wires per link	16

TABLE I  
PARAMETERS FOR THE PROCESSOR CHIP AREA MODEL.

Parameter	Value
Wire delay	125ps/mm
Clock rate	1GHz
Latency tile-to-switch	1 cycle
Latency switch	2 cycles
Latency switch when route closed	5 cycles
Serialisation latency	2 cycles
Link latency	2 cycles

TABLE II  
PARAMETERS FOR THE CHIP DELAY MODEL.

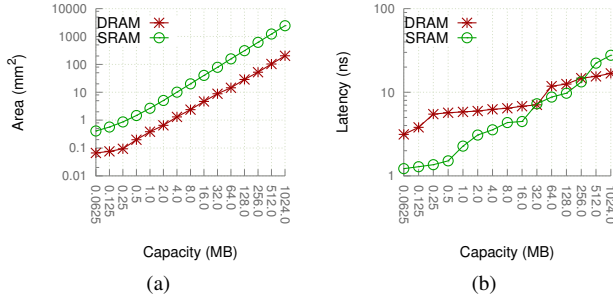


Fig. 6. Log-log plots of CACTI estimates for area and access latency for commodity 40nm DRAM as a function of capacity. 28nm SRAM embedded on a logic process is included as a baseline.

horizontal connections in one metal layer but each other switch group makes branching connections from both metal layers, which restricts both the horizontal and vertical placement of switches. Example arrangements of both groups are illustrated for the 256-tile example in Fig. 5b. The resulting width of the wiring channels is determined by that of the child switch group and the area of the interconnect is calculated as the sum of the area of the wiring channels and the switch groups.

A 2D mesh network is laid out as an array of blocks of processors where each block of processors attaches to a single switch. These are separated by wiring channels accommodating the width and height of a switch, which is placed at the corner of its block. Adjacent horizontal and vertical links connect directly between switches. The layout for a 256 tile 2D mesh network is given in Fig. 5c.

The decision to route interconnect wires in dedicated channels simplifies the model but area estimates will be generous for the Clos network in particular, as in principle interconnect wires can be routed over any other resources on the chip, with a small area overhead from repeaters and buffers. Finally, it is assumed that connections between switches to processors can be routed over other resources.

## B. Area parameters

The area of the XCore processor on a 90nm process is estimated to be 1mm<sup>2</sup> and, applying a linear scaling, the area on a 28nm process is estimated to be 0.10mm<sup>2</sup>. The area of the switch is estimated by scaling the 32×32 C104 switch, which occupied 40mm<sup>2</sup> on a 1μm process, to be 0.03mm<sup>2</sup> on 28nm. These figures are consistent with the ARM Cortex-M0 processor [40] and the 32×32 SWIFT switch [28]. The Cortex-M0 is a simpler processor which has a 3-stage pipeline and supports a single hardware thread. On a 40nm process it has an area of 0.01mm<sup>2</sup> and on 28nm an estimated area of 0.003mm<sup>2</sup>. The SWIFT switch has an area of 0.35mm<sup>2</sup> on a 65nm process and an estimated area of 0.06mm<sup>2</sup> in 28nm. All links are 8 bits wide and require 16 wires, 8 in each direction. The pitch of wires in semi-global metal layers for 28nm is 125nm [41, Tab INTC6].

The area of commodity DRAM on a 40nm process is estimated with the CACTI tool [42] and Fig. 6a shows the computed values and includes the corresponding figures for SRAM embedded in 28nm logic as a baseline. For small DRAMs, the area becomes dominated by the peripheral circuitry and interconnect and there is little reduction in area below 0.25MB. Even at this point, SRAM is still 5 to 7 times less dense and thus not practical for building systems with large amounts of memory. Fine pitch TSVs are around 50μm which allows 400 connections per square millimeter.

## C. Network performance

In contrast to parallel programs, execution of sequential programs will not induce any concurrent traffic in the network and unless additional processes are run in parallel, each message will travel without contention. In this case, which is assumed for the evaluation, the interconnect needs only to provide low latency and high bandwidth at zero-load<sup>2</sup>. The following model of latency is based on, and calibrated against measurements made with a real XMOS device [43]. The zero-load latency ( $t_0$ ) of a message sent from processor  $s$  to processor  $d$  depends on the latency of link between the tile and the switch ( $t_{TS}$ ); the switch latency when a route is open ( $t_S$ ), and additional switch latency if the route is closed ( $t_{SC}$ ); the length of the path ( $h(s, d)$ ); the link latency ( $t_L$ ); and the serialisation latency ( $t_{SR}$ ) which is determined by the message length and channel bandwidth.  $t_0$  is calculated as

$$2t_{TS} + t_{SR} + (h(s, d) + 1)(t_S + t_{SC}) + h(s, d)t_L$$

if the route is closed, and

$$2t_{TS} + t_{SR} + (h(s, d) + 1)t_S + h(s, d)t_L$$

if the route is open. Shortest path routing is assumed so  $h(s, d)$  is the minimum distance between  $s$  and  $d$ . Values for the latency parameters are summarised in Tab. II.

<sup>2</sup>Depending on the system requirements, it could be beneficial to provision a complete separate network for memory traffic so it is not disrupted by other potentially latency-insensitive traffic. The Tiler and Adaptea architectures both do this.

A processor and interconnect clock rate of 1GHz is chosen as this is realistic for a system with large numbers of processors and it allows the memory access latencies to be normalised to cycles. A wire delay of 125ps/mm is characteristic for standard repeated wires at a 125nm pitch and, at 1GHz, signals can travel along these at 8mm/cycle. Based on the VLSI layout, the propagation delay never exceeds 2 cycles for any single wire and this is fixed for all wire delays. The DRAM access latency was also estimated with CACTI and Fig. 6b shows the computed values. The performance of the PM is simulated with a modified version of AXE<sup>3</sup> to incorporate the latency model for communications and local memory accesses.

## V. EVALUATION OF THE EMULATION

Performance of the PM emulation of a SM is judged by the relative *slowdown* of the emulation compared to the SM running a sequential program. Implementation cost is judged by comparing how the requirements of processing and interconnect compare with memory, which does not yield precise comparisons but is sufficient to gain a general idea of relative costs. The evaluation focuses on how both aspects scale with the number of tiles.

Based on the chip model and parameters discussed, systems with 64 to 4096 tiles are constructed and a range of memory capacities from 1GB to 8GB are considered. These are intended to be representative of typical DRAM systems and this allows the performance of the PM to be compared against a model of a DRAM system that captures sources of latency such as the memory controller and bus protocols. This model does not capture wire delay or their packaging in a DIMM so it leaves potential for a stacked package as is proposed for the PM.

### A. Implementation cost and scaling

Fig. 7a shows how the total area of the processors and interconnect components of the system scale with the number of tiles. The area of the Clos network increases more quickly than the mesh. At 2048 tiles, the Clos occupies about 50% more than the mesh, and at 4096 tiles, 70% more. This is not surprising given the simple floorplan. At this size, the switch groups will occupy a significant area. For 4096 tiles, there is potential for significant optimisation of the layout to reduce wasted area in the switch groups and wiring channels. Overall, the implementation cost of the Clos is higher than the mesh and scales well for all but the largest networks. Fig. 7b shows the total area of tile memories for various memory capacities and highlights the overhead associated with splitting memories into larger groups of smaller capacities. This is due to the peripheral circuitry and interconnect associated with each memory. The increases are also not linear and the step changes are due to characteristics of the DRAM layouts.

Fig. 8 shows the area of the processor, switch and wiring components of the processing chip per tile. In the mesh, there is a constant overhead for all of these components per tile and

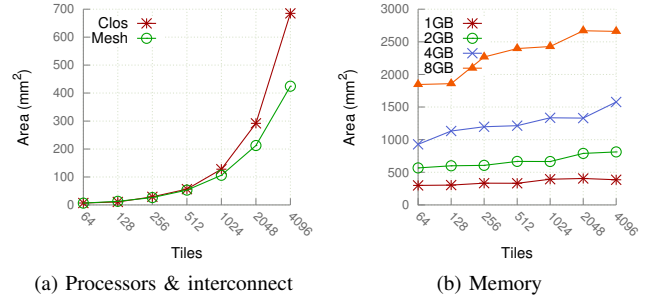


Fig. 7. Log-linear plots of total area requirements as a function of the numbers of tiles for (a) processors and interconnect and (b) the total area occupied by tile memories.

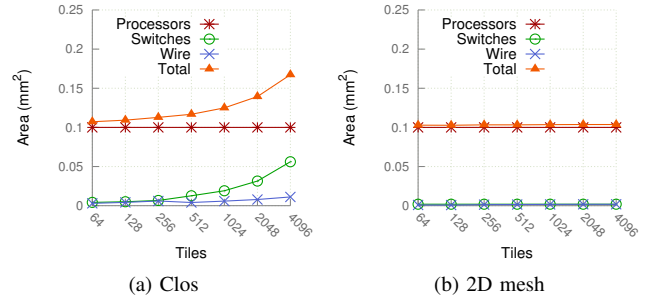


Fig. 8. Log-linear plots of area per tile with a breakdown of the switch, wire, and processor components.

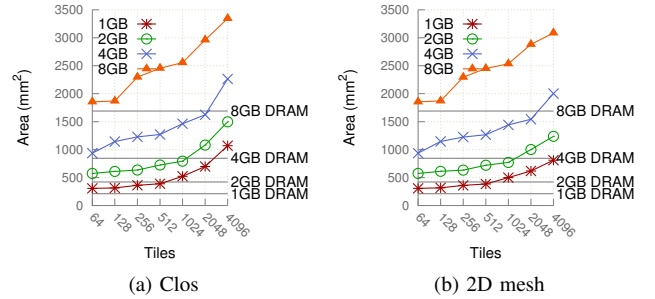


Fig. 9. Log-linear plots of the total silicon area occupied by the processors, interconnect and memories of systems, compared to the area of monolithic DRAMs, for different total memory capacities.

this is clearly shown in Fig. 8b. The total processing overhead is  $0.10\text{mm}^2$  per tile and this is dominated by the area of the processors; the switch and wire area is less than  $0.01\text{mm}^2$ . In the Clos, the area of the processors dominates the total area up to 2048 tiles. After this, the switch area becomes a significant proportion. This is because it is the total switch group area, which is larger than just the sum of the switch areas. The wire area per tile increases slightly with the number of tiles as it is based on the total area occupied by the dedicated channels, but remains relatively low throughout. With 64 tiles it accounts for 5% of the total area and at 4096, just 7.5%.

Fig. 9 shows the total area occupied by each system for various memory capacities. Horizontal lines are drawn to mark the area of various sizes of monolithic DRAMs for comparison, which are estimated with CACTI. The area of the SM would also include the processor and cache but these are not included as they are negligible: the processor occupies  $0.10\text{mm}^2$  and a typical 8MB SRAM on a 28nm logic process

<sup>3</sup>An XCore Emulator; the original version of AXE is available from [http://github.com/rlosborne/tool\\_axe](http://github.com/rlosborne/tool_axe) and the version developed for this work is available from [http://github.com/jameshanlon/tool\\_axe](http://github.com/jameshanlon/tool_axe).

occupies  $20.11\text{mm}^2$ . Overall, the area of the PM up to 4096 tiles is to within a factor of 2 of the SM, even with the conservative estimate of SM area. With memories included, the difference between the total area of Clos and mesh systems is around 10% representing only a small cost for the benefits obtained in performance.

## B. Performance

1) *Methodology*: The SM is modelled as a single processor attached to a DRAM memory as depicted in Fig. 1b. The processor is the same as the ones used in the PM. For fairness of comparison the SM is modelled using the assumption that memory accesses to the areas that are stored in local memory in the PM incur the same latency. The remainder of global accesses incur a fixed latency, based on the performance of a DRAM system. This assumption captures the effect of a cache but is optimistic as global memory accesses typically constitute 10% to 20% of executed instructions. The effect is similar to providing the SM with a fast cache memory with an 80% to 90% hit rate.

The access latency is estimated for a modern DRAM by simulation with DRAMSim2 [44]. Performance is measured by performing read and write accesses to addresses chosen uniformly at random over the address range and the fixed latency is calculated as the average of these accesses. Accesses are issued only once the last has completed to restrict the memory controller to processing a single transaction at a time. For a system with 1Gb DDR3 chips [45], average random access latency is measured at 35ns for a single rank of 1GB capacity. For multi-rank system with 2GB to 16GB capacities, this increases to 36ns due to a small overhead in switching between ranks. As DRAMSim2 does not model a particular packaging or wire delay, the figures are inclusive of stacked DRAM integrated with TSVs.

The total memory capacity is held constant at 4GB. This choice is not important as it makes only a small difference to access latencies, compared to other capacities in the 1 to 16GB range (less than 10ns) and little difference in the overall performance of both models; the empirical analysis is concerned with obtaining results to within small factors and to demonstrate scaling behaviour.

The analysis is composed of two parts. The first uses synthetic instructions sequences to characterise conventional sequential programs. These contain a certain ratio of global memory accesses to local memory and non-memory operations. Execution of the Dhrystone [46] general-purpose sequential benchmark was analysed to determine the proportions of access types to produce a synthetic version. Additional variations of instruction mixes are included to observe the effect on performance. The second part uses a compiler as a case study example of a realistic general-purpose application.

2) *Absolute latency*: Fig. 10 shows the average access latency of random reads and writes in the emulated memory and the baseline latency measured from the simulated DDR3 memory. Latency in the mesh increases linearly with the number of tiles, to around 220ns at 4096 tiles, 6.3 times that of the DRAM latency. The results for the Clos network clearly

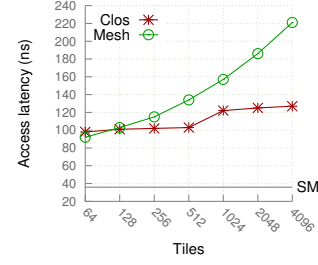


Fig. 10. Absolute access latency of the emulated memory for both networks. The horizontal line shows the latency of the SM that was measured with DRAMSim2.

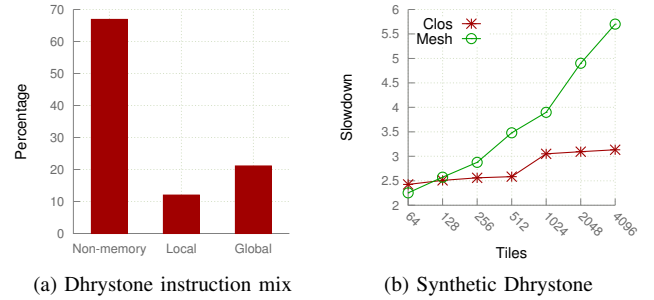


Fig. 11. The mix of instructions in the Dhrystone benchmark and the performance slowdown of the synthetic benchmark with the same instruction mix, relative to the SM.

reflect the logarithmic growth of the network diameter and show the extra latency incurred by additional stage in the 1024 to 4096 tile systems. As expected, the access latency scales well with system size, up to 125ns at 4096 tiles, 3.6 times that of the DRAM latency.

3) *Synthetic benchmarks*: Fig. 11 presents the instruction mix of the Dhrystone benchmark and the performance of a synthetic version with the same mix. The general behaviour reflects that of Fig. 10 but with a mix of emulated global accesses, local accesses and non-memory operations, the overhead is lower than in the direct comparison of access latencies. For the Clos, it achieves a slowdown of up to about 2.6 over the SM. The mesh performs better at 64 tiles due to average shorter paths in the network but this scales up to a slowdown of around 5 times at 4096 tiles. In general, as the ratio of global memory operations to local and non-memory operations decreases, the slowdown does also. As the proportion of global accesses increases, performance converges to the worst case slowdown, the ratio between the PM and SM as illustrated in Fig. 10. Figures 12a and 12b show results for various ratios of global memory accesses with the synthetic benchmark. In these, the proportion of local memory access is fixed at 10%, based on the observation with the Dhrystone benchmark.

4) *Compiler case study*: To study the application of the proposed scheme for executing sequential programs and its performance in a realistic general-purpose application, a bootstrapping compiler is used and modified to generate code that performs memory accesses to data structures through message sequences to communicate with an emulated memory. It can be modified to use the mechanism itself to allow analysis of its performance compared to the version with conventional



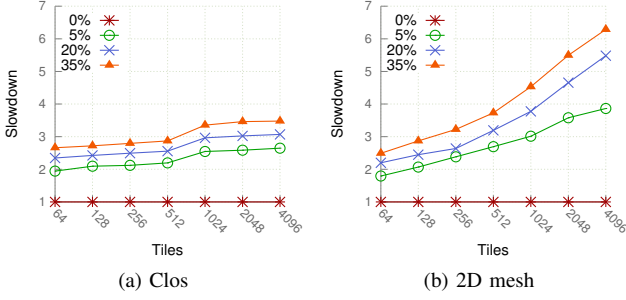


Fig. 12. Performance slowdown of the synthetic benchmark relative to the SM with various proportions of global memory accesses in all executed operations.

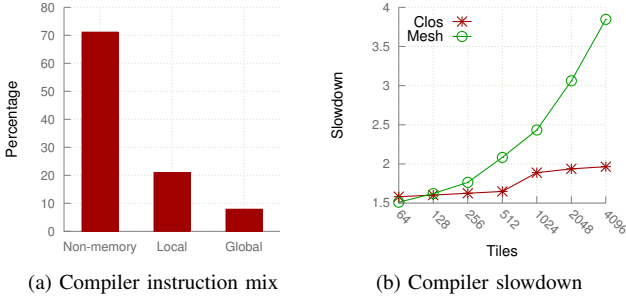


Fig. 13. The mix of instructions in the compiler and its performance slowdown relative to the SM when self-compiling.

memory accesses. The largest data structures in it are a parse tree, name table, label table and instruction buffer, and these were modified to use the emulated memory. To bootstrap itself, the parse tree and name table require 70,000 words, the label table 4,000 and the code buffer 20,000.

Fig. 13a shows the instruction mix of the original version of the compiler. It is based on a recursive descent traversal of the parse tree so the execution is highly recursive and the proportion of local memory accesses are dominated by access to the stack and accordingly high (21%) compared to the Dhrystone instruction mix (12%). As each memory reference is written as a sequence of communications, the resulting size of the program binary is also larger. For the compiler, the increase was 8%, but this is a small price compared to the amount of extra memory that can be provided, and there would be no overhead if the memory emulation was supported in hardware. Using current X MOS processors with a word size of 4 bytes and 64KB memories, a total of 376KB storage is required, which can be provided by 6 tiles.

Fig. 13b presents slowdown results of self-compilation for much larger systems to demonstrate how this scales. Since the proportion of global accesses is lower than Dhrystone, the slowdown is also, just 1.5 times for both networks at 64 tiles and up to 1.9 and 4.0 times slowdown for the Clos and mesh respectively at 4096 tiles. What these results emphasise is that the choice of how much memory to provision to each processor is no longer important. This makes the system much more flexible for only a marginal cost in latency.

5) *Summary*: Overall, for general sequential programs where there is a mix of operations, although the absolute

access latency into the emulated memory is high (a factor of 3 to 4 times that of the SM), the effect of this is diluted by fast local accesses and other non-memory operations. The results also show that performance of the two systems is roughly comparable up to 256 tiles (a switch-hop diameter of 4), and this suggests that it may be practical to use a mesh for systems up to this size with high-degree switches; or equally that this approach could be applied to existing mesh-based systems. Beyond this point, and up to 4096 tiles where the network diameter is 16, the performance of the mesh degrades as communication latency increases linearly. In contrast, the performance of Clos network scales well and it maintains an almost constant slowdown. Even for a program with 30% global memory accesses, the emulation can deliver performance to within a factor of 3.5 of the SM with 4096 tiles. In practice, 10% to 20% of operations in general-purpose codes are global memory access and these programs can be emulated with only a small factor of around 2 to 3 slowdown.

## VI. DISCUSSION

### A. Engineering optimisations

The evaluation uses latency as the performance metric and simplified models of the two systems capture the essential aspects of this. A modern DRAM employs a number of engineering optimisations to scale bandwidth, which were not considered in the evaluation, but can in principle be applied to the PM. For example, double data rate (DDR) DRAMs allow two data items to be transferred every cycle, one on each edge of the clock. This is achieved by replicating a memory and parallelising access to it. DDR DRAMs could be used for local memories and at the system-level a similar effect could be achieved by increasing the frequency of the interconnect. Also, there is a cost associated with accessing a row in a DRAM and successive reads from an open page will exhibit lower latency. Memory controllers can issue commands based on address, scheduling and queueing policies and dynamically reorder outstanding transactions to best exploit this. Additional functionality like this would be straight-forward to implement in the emulation's memory controller.

Optimisations could also be applied to the PM to reduce the latency overhead associated with request and reply messages in accesses to the emulated memory. Firstly, the scheme described in this paper employs a software memory controller. This functionality could be implemented in hardware and even as part of the ISA so that load and store instructions accessing particular address ranges are converted into messages and sent on the interconnect automatically. Secondly, the latency model was calibrated against measurements made with a real X MOS device, but the switch implementation in this is automatically synthesised which potentially adds a factor of 2 to 3 times the latency in the critical path over a manual layout. This could reduce the switch latency to 1 cycle and overhead to open a route to 2 cycles.

### B. Scaling

The chip layout for the Clos network presented in Section IV-A is effective up to 2048 tiles, but for the larger

4096 tile network, the area of the switch groups dominates the system area. Theoretical results for VLSI layouts of fat tree variants [47] may be applied to obtain a more compact layout of the Clos networks studied and the use of additional metal layers for global interconnect wires would reduce wiring congestion [48].

It would also be natural to scale the system by connecting multiple chips with additional external stages of the Clos network since links can be taken directly off-chip from the core switches. In practice though, this is prohibitively difficult with current technology. Electrical connections on chip packages are very limited. A high density package might have around 800 pins<sup>4</sup> and typically around 40% of these have to be used for ground and power, leaving only 480 for signals. For bidirectional links with 8-bits in each direction, only 30 such links could be connected. Such a package would also require 4 to 6 wiring layers on a printed circuit board (PCB) to route tracks to each of the pins. Even with the links on a PCB from a chip, with a wiring pitch of 0.15mm, routing large sets of wires requires a large amount of area, for example 256 wires require a minimum width of around 40mm. It would be possible to connect chips together in a lower dimensional network such as a mesh but communication latency would increase accordingly when moving between chips.

Several emerging technologies could significantly improve the packaging density and power consumption for large systems with multiple chips, and allow large numbers of connections between chips. Optical interconnections can potentially be made with a silicon chip [50] and would allow a large number of signals at high bandwidth to be carried down a single optical fibre. Silicon interposers [51] allow chips to be mounted and connected with TSVs to a silicon substrate to provide high connectivity. This is already a production technology used in Xilinx FPGAs [52].

### C. Extensions

With the proposed scheme for execution of sequential programs, there is potential to further exploit the underlying architecture. The following points outline several ideas.

- With a processor associated with, and mediating access to, each memory in the system, data read and written in remote memories could be processed on-the-fly. In particular, compression could be applied before sending messages to increase throughput and reduce latency.
- Access latency could be reduced moving processes to the processor storing particular data to operate on it in-place in local memory, rather than transferring it back and forth.
- Unlike a conventional DRAM, an emulated memory can support concurrent access. A sequence of reads from distinct locations, which might occur as terms in an expression, could be issued simultaneously. The effective cost of this sequence would then be similar to that of a read. This is a simple optimisation that could be applied during compilation to sequential programs.

- When unused or inactive, individual memories could be switched off or placed in a low-power state by their associated processor to reduce power consumption. This could also be applied to the processor.
- As all memory requests are issued as messages sent on the interconnect, they could be intercepted to provide debugging or profiling information.
- Multiple instances of the sequential emulation scheme could be run in parallel to emulate a parallel machine with a larger memory capacity per tile, effectively reducing the granularity of the system.

It is also worth noting that this scheme can in principle be applied to any system where it is possible to access the memory of any other processor, such as the Adaptive Epiphany and Intel SCC.

## VII. CONCLUSION

This paper presents a tiled parallel architecture that can scale to thousands of processors per-chip. As well as executing highly parallel programs, it can deliver an efficient emulation of a large memory using collections of smaller ones, allowing it to support sequential execution. This is efficient both in terms of a low overhead in implementation cost and a small slowdown in performance when compared with a sequential machine. Architecturally, it is similar to a DRAM system as a collection of memories connected by an interconnection network. In a DRAM system, the interconnect is specialised to transmit particular access requests and memory data, and in the parallel system it supports concurrent message passing traffic. With a Clos network, the proposed parallel machine can deliver an efficient emulation of a sequential machine by providing scalable low-latency communications. By using a high-degree switches it is practical to construct large Clos networks with a small diameter. With  $32 \times 32$  crossbar switches, 4096 tiles can be connected with a maximum distance of 4 switch hops between any pair of tiles, compared with 16 in a mesh.

Using a detailed VLSI model for an implementation of the system on-chip, operation throughput and implementation cost were evaluated. Implementation cost was considered in order to demonstrate that as well as the parallel system being practical to build, the area it occupies is around a factor of 2 of the sequential machine it is emulating. Synthetic benchmarks were used to characterise conventional general-purpose sequential programs and explore the impact of varying proportions of global memory accesses. For realistic general-purpose applications, where 10% to 20% of executed operations are global memory accesses, the proposed parallel system can deliver a highly parallel emulation (up to 4096 cores) with an overall slowdown of 2 to 3 when compared to the specialised implementation. The result of this is an architecture that can switch from executing parallel programs with thousands of processes to sequential programs with large memory requirements. The choice of how much memory to provision per tile in this system is not important, making it very flexible for both the system designer and programmer.

<sup>4</sup>This is a lot fewer than the number of available chip I/O pads, which for a typical microprocessor is up to 3367 [49, Tab. ORTC-4].

## ACKNOWLEDGMENT

This work was funded by EPSRC grant SB1933.

## REFERENCES

- [1] S. H. Fuller and L. I. Miller, Eds., *The Future of Computing Performance: Game Over or Next Level?* National Academies Press, 2011.
- [2] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 746–749.
- [3] International Technology Roadmap for Semiconductors, "System drivers, 2011 edition," 2011.
- [4] Intel, "Many Integrated Core (MIC) Architecture," 2012, <http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html>.
- [5] Howard et al., "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, February 2010, pp. 108–109.
- [6] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *Micro, IEEE*, vol. 27, no. 5, pp. 15–31, September 2007.
- [7] Haring et al., "The IBM Blue Gene/Q Compute Chip," *Micro, IEEE*, vol. 32, no. 2, pp. 48–60, March–April 2012.
- [8] "big.LITTLE Processing," ARM Ltd., 2012, <http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php>.
- [9] "AMD Accelerated Processing Units," Advanced Micro Devices Inc., 2012, <http://www.amd.com/us/products/technologies/fusion/Pages/fusion.aspx>.
- [10] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to software: Raw machines," *Computer*, vol. 30, no. 9, pp. 86–93, September 1997.
- [11] A. Inc., "1024-core 70GFLOP/W floating point manycore microprocessor," October 2011, whitepaper.
- [12] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: a modular reconfigurable architecture," *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 161–171, 2000.
- [13] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, October 1985.
- [14] L. G. Valiant, "General purpose parallel architectures," in *Handbook of theoretical computer science (vol. A): algorithms and complexity*. Cambridge, MA, USA: MIT Press, 1990, pp. 943–973.
- [15] F. T. Leighton, *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.
- [16] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, D. Truong, T. Mohsenin, and B. Baas, "Asap: An asynchronous array of simple processors," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 3, pp. 695–705, March 2008.
- [17] G. Panesar, D. Towner, A. Duller, A. Gray, and W. Robbins, "Deterministic parallel processing," *Int. J. Parallel Program.*, vol. 34, no. 4, pp. 323–341, August 2006.
- [18] INMOS Ltd., *Transputer Databook*, INMOS Ltd., 1988, first Edition.
- [19] D. May, *The XMOS XS1 Architecture*, XMOS Ltd., October 2009.
- [20] M. D. May, P. W. Thompson, and P. H. Welch, Eds., *Networks, Routers and Transputers: Function, Performance and Applications*, 1st ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1993.
- [21] D. May, "Communicating process architecture for multicores," *Concurrency and Computation: Practice and Experience*, vol. 22, pp. 935–948, June 2010.
- [22] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, March 1953.
- [23] C. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong-chan, S. wen Yang, and R. Zak, "The network architecture of the Connection Machine CM-5," in *Journal of Parallel and Distributed Computing*, 1992, pp. 272–285.
- [24] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho, "Entering the petaflop era: The architecture and performance of Roadrunner," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, November 2008.
- [25] SGI, "Technical advances in the SGI UV architecture," 2011, white paper.
- [26] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74.
- [27] S. Scott, D. Abts, J. Kim, and W. J. Dally, "The BlackWidow high-radix Clos network," in *Proceedings of the 33rd annual international symposium on Computer Architecture*, ser. ISCA '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 16–28.
- [28] S. Satpathy, R. Dreslinski, D. S. T. Ou, T. Mudge, and D. Blaauw, "SWIFT: A 2.1Tb/s 32x32 self-arbitrating manycore interconnect fabric," ser. Symposia on VLSI Technology and Circuits, Koyoto, Japan, June 2011, pp. 138–139.
- [29] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proceedings of the 20th annual international conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 187–198.
- [30] D. Ludovici, F. Gilabert, S. Medardoni, C. Gómez, M. E. Gómez, P. López, G. N. Gaydadjiev, and D. Bertozzi, "Assessing fat-tree topologies for regular network-on-chip design under nanoscale technology constraints," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 562–565.
- [31] Y.-H. Kao, M. Yang, N. Artan, and H. Chao, "Cnoc: High-radix clos network-on-chip," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 12, pp. 1897–1910, December 2011.
- [32] A. Joshi, C. Batten, Y.-J. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanovic, "Silicon-photonics clos networks for global on-chip communication," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–133.
- [33] J. Zhang, H. Gu, and Y. Yang, "A high performance optical network on chip based on Clos topology," in *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, vol. 2, May 2010, pp. V2–63–V2–68.
- [34] Tezzaron Semiconductor, "Octopus 8-Port DRAM for Die-Stack Applications," 2010, [http://tezzaron.com/memory/datasheets/TSC10080x\\_0\\_1.pdf](http://tezzaron.com/memory/datasheets/TSC10080x_0_1.pdf).
- [35] JEDEC, "Wide I/O single data rate (Wide I/O SDR)," December 2011, jESD229.
- [36] "Hybrid Memory Cube Consortium," 2012, <http://hybridmemorycube.org/>.
- [37] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.
- [38] D. A. Patterson, "Latency lags bandwidth," *Commun. ACM*, vol. 47, pp. 71–75, October 2004.
- [39] A. M. Jones, N. J. Davies, M. A. Firth, and W. C. J., *The Network Designer's Handbook*, 1st ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1997.
- [40] A. Ltd., "Cortex-M0 Processor," 2012, <http://www.arm.com/products/processors/cortex-m/cortex-m0.php>.
- [41] International Technology Roadmap for Semiconductors, "Interconnect," 2010.
- [42] S. Thoziyoor, N. Muralimanohar, J. Ho Ahn, and N. P. Jouppi, "Cacti 5.1," HP Laboratories, Tech. Rep., 2008.
- [43] XMOS, *XK-XMP-64 Hardware Manual*, XMOS Ltd., February 2010.
- [44] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, January 2011.
- [45] Micron, "1Gb: x4, x8, x16 DDR3 SDRAM: MT41J128M8JP-125 Features," <http://www.micron.com/products/dram/ddr3-sdram/ddr3-sdram-part-catalog>.
- [46] R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," *Commun. ACM*, vol. 27, no. 10, pp. 1013–1030, October 1984.
- [47] R. I. Greenberg and C. E. Leiserson, "A compact layout for the three-dimensional tree of meshes," *Applied Mathematics Letters*, vol. 1, no. 2, pp. 171–176, 1988.
- [48] A. DeHon, "Compact, multilayer layout for butterfly fat-tree," in *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '00. New York, NY, USA: ACM, 2000, pp. 206–215.
- [49] International Technology Roadmap for Semiconductors, "Executive summary, 2011 edition," 2011.
- [50] D. A. B. Miller, "Optical interconnects to electronic chips," *Appl. Opt.*, vol. 49, no. 25, pp. F59–F70, September 2010.

- [51] M. Sunohara, T. Tokunaga, T. Kurihara, and M. Higashi, "Silicon interposer with TSVs (through silicon vias) and fine multilayer wiring," in *Electronic Components and Technology Conference, 2008. ECTC 2008. 58th*, may 2008, pp. 847 –852.
- [52] Saban, Kirk, "Xilinx stacked silicon interconnect technology delivers breakthrough FPGA capacity, bandwidth, and power efficiency," 2011.